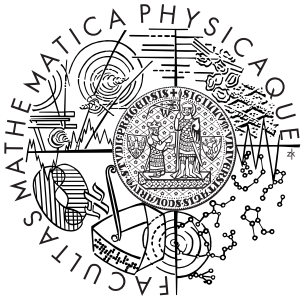
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁRSKA PRÁCA



Rudolf Tomori

Automatická detekcia zraniteľností vo webových aplikáciach

Katedra softwarového inženýrství

Vedúci bakalárskej práce: RNDr. Pavel Parízek, Ph.D.

Študijný program: Informatika, správa počítačových systémov

2009

Na tomto mieste by som rád poďakoval svojmu vedúcemu ročníkového projektu a bakalárskej práce RNDr. Pavlovi Parízkovi, Ph.D. za jeho cenné rady a pripomienky pri tvorbe tejto práce. Ďalej by som chcel poďakovať svojim rodičom za to, že mi umožnili plne sa sústrediť na štúdium a prípravu tejto práce.

Prehlasujem, že som svoju bakalársku prácu napísal samostatne a výhradne s použitím citovaných prameňov. Súhlasím so zapožičaním práce a jej zverejnením.

V Prahe dňa 21.5.2009 Rudolf Tomori

Obsah

Názov práce: Automatická detekcia zraniteľností vo webových aplikáciach
Autor: Rudolf Tomori
Katedra (ústav): Katedra softwarového inženýrství
Vedúci bakalárskej práce: RNDr. Pavel Parízek, Ph.D.
e-mail vedúceho: Pavel.Parizek@mff.cuni.cz

Abstrakt: Významným a častým problémom webových aplikácií sú bezpečnostné chyby, z ktorých veľká časť sa dá pomerne presne deterministickým algoritmom identifikovať pomocou na to určeného špecializovaného softwaru. Nástroj *WebCop* vyvinutý v rámci tejto práce dokáže lokalizovať vo webových aplikáciách vybrané bezpečnostné zraniteľnosti. Jeho výhoda oproti bežne dostupnému softwaru spočíva v konfigurovateľnosti prevádzaných testov, kde samotný užívateľ definuje podmienky určujúce prítomnosť konkrétnej zraniteľnosti. Nástroj je implementovaný v jazyku C++ a je určený pre platformu Unix/Linux.

Kľúčové slová: bezpečnosť webových aplikácii, XSS, CSRF, SQL injection

Title: Automatic detection of web application vulnerabilities
Author: Rudolf Tomori
Department: Department of Software Engineering
Supervisor: RNDr. Pavel Parízek, Ph.D.
Supervisor's e-mail address: Pavel.Parizek@mff.cuni.cz

Abstract: Security vulnerabilities, while being an important and common problem of web applications, are often easily detectable by using specialised software. The *WebCop* tool developed in this thesis is able to spot some of the common web application security vulnerabilities. Its main advantage over other similar software lies in its ability to configure the way in which the individual tests are performed, also allowing the user to specify the conditions that are required to be met in order to acknoledge the presence of the given security vulnerability. The tool is developed for the Unix/Linux platform using the C++ language.

Keywords: web application vulnerability, XSS, CSRF, SQL injection.

Kapitola 1

Úvod

Dôležitou vlastnosťou webových aplikácií, ktorá by mala byť braná na zreteľ už v dobe návrhu, je bezpečnosť. Žiaľ dnes je tento pojem v súvislosti s webovými aplikáciami často podceňovaný. To môže mať vážne priame či nepriame dôsledky, ktorých pravdepodobnosť je zvyšovaná vysokým výskytom najrôznejších webových služieb. Medzi typické následky patrí predovšetkým získanie dôverných informácií tretími stranami, prípadne neoprávnené využitie prostriedkov cez zraniteľné webové rozhranie.

Témou tejto práce je popis nástroja *WebCop* na black-box testovanie bezpečnosti webových aplikácií. To znamená testovanie aplikácie bez znalosti jej zdrojového kódu, iba na základe reakcií na špeciálne zvolené vstupné data. Cieľom tohto nástroja je pomôcť webovým vývojárom zautomatizovať a tým zjednodušiť testovanie ich aplikácií na niekoľko vybraných základných známych typov bezpečnostných zraniteľností.

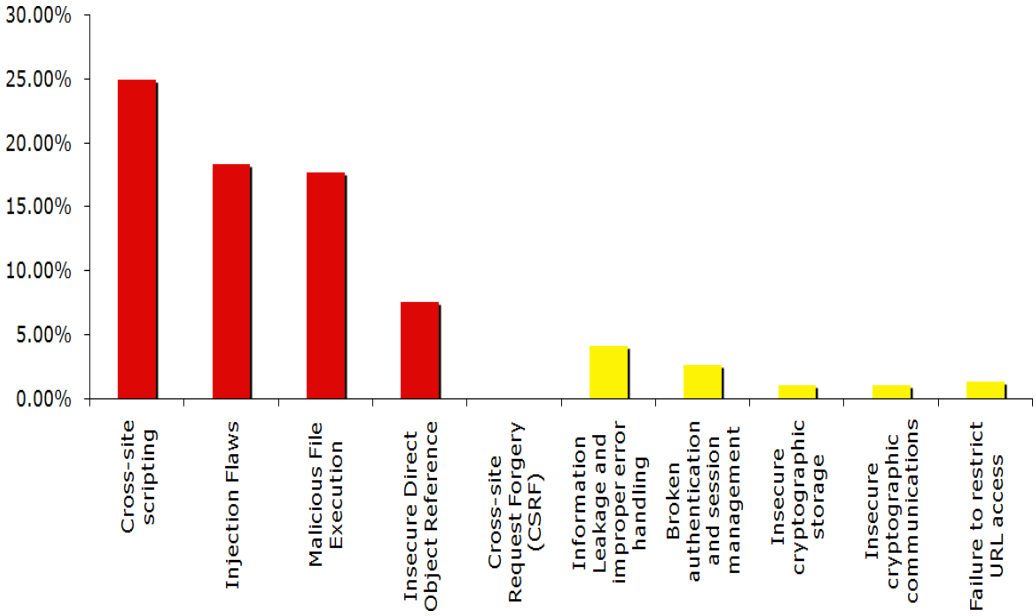
Nástroj *WebCop* teraz podporuje šesť typov webových zraniteľností, ale je možné ho ďalej rozšíriť o detekciu ďalších zraniteľností. Konkrétne obsahuje podporu pre tieto zraniteľnosti:

1. Cross-site scripting
2. SQL injection
3. Cross-site Request Forgery
4. Remote file inclusion

5. Local file inclusion

6. Upload vulnerabilities

To že sa tieto zraniteľnosti vo webových aplikáciách skutočne vyskytujú môžem doložiť štatistikou z projektu *OWASP Top Ten Project* [2], ktorý sa snaží identifikovať 10 najvýznamnejších bezpečnostných zraniteľností webových aplikácií.



Obr. 1.1: Desať najvýznamnejších bezpečnostných problémov webových aplikácií

Tento stĺpcový graf zobrazuje percentuálne zastúpenie jednotlivých typov zraniteľností medzi verejne známymi bezpečnostnými zraniteľnosťami zozbieranými v databáze CVE spravovanou spoločnosťou *The MITRE Corporation*. Zraniteľnosťi SQL Injection sú súčasťou stĺpca *Injection Flaws*, zraniteľnosť Local File Inclusion je súčasťou stĺpca *Insecure Direct Object Reference*, Remote File Inclusion a Upload vulnerabilities sú súčasťou stĺpca *Malicious File Execution*.

Táto štatistika bola zostavená z dát *Mitre Vulnerability Trends* [3] pre rok 2006, z ktorých sa vybralo 10 najvýznamnejších bezpečnostných problémov

webových aplikácií. Červená farba stĺpca značí, že daná zraniteľnosť patrí do prvej desiatky z hľadiska počtu všetkých zraniteľností nahlásených v príslušný rok. Žltá farba značí, že táto zraniteľnosť bola aspoň 5% nad priemerom. V tejto top desiatke sa nachádza aj *CSRF*, čo autori projektu *OWASP Top Ten* oddôvodňujú tým, že skutočný výskyt tejto zraniteľnosti je vyšší, ako ukazuje jej ranking [2].

Text tejto práce je rozčlenený do siedmych kapitol. Prvú kapitolu tvorí úvod. V druhej kapitole stručne predstavujem podporované typy bezpečnostných zraniteľností. Tretia kapitola ponúka analýzu problému automatizovaného testovania webových aplikácií na bezpečnostné zraniteľnosti. Vo štvrtej kapitole popisujem programátorskú dokumentáciu nástroja *WebCop*. Piata kapitola tvorí užívateľskú dokumentáciu, vrátane podrobného popisu konfigurácie nástroja. V šiestej kapitole sa nachádza porovnanie kladov a záporov nástroja *WebCop* a iných dostupných podobných nástrojov. V siedmej kapitole ponúkam záverečné zhrnutie.

Kapitola 2

Podporované zraniteľnosti

2.1 Cross Site Scripting

> XSS is the New Buffer Overflow, JavaScript Malware is the New Shell Code.
>
> Jeremiah Grossman

Zraniteľnosti *Cross Site Scripting* alebo *XSS* sa nachádzajú vo webových aplikáciach, ktoré umožňujú útočníkovi špeciálnym zvolením vstupných dát manipulovať s client-side kódom webovej aplikácie. Tým je užívateľovi umožnené manipulovanie s *DOM* webovej aplikácie, útočník ma tiež k dispozícii klientské skriptovacie jazyky, ako napríklad *JavaScript*. To môže mať podľa povahy *XSS* za následok zmenu vzhľadu webovej stránky, ukradnutie autentifikačných cookies, prípadne vykonanie pokročilejších útokov pomocou XSS-Proxy.

Podľa *OWASP Testing Guide* [1] rozoznávame 4 typy zraniteľností XSS.

1. Reflected Cross Site Scripting (OWASP-DV-001)
2. Stored Cross Site Scripting (OWASP-DV-002)
3. DOM based Cross Site Scripting (OWASP-DV-003)
4. Cross Site Flashing (OWASP-DV-004)

Nástroj *WebCop* umožňuje detekciu zraniteľností *Reflected Cross Site Scripting*, v niektorých prípadoch aj zraniteľností *Stored Cross Site Scripting* a *DOM based Cross Site Scripting*.

2.1.1 Reflected Cross Site Scripting

Tento typ zraniteľnosti je známy aj ako neprezistentné XSS. Ide o prípady, kedy sa útočníkov kód nespúšťa vždy pri načítaní zraniteľnej aplikácie, ale je spôsobený tým, že obeť spustila aplikáciu pomocou útočníkom podstrčenej URL adresy.

2.1.2 Stored Cross Site Scripting

Táto zraniteľnosť nastáva, keď webová aplikácia zbiera užívateľské data, nefiltruje ich korektne a tieto data ukladá pre neskoršie použitie. Ako následok sa tieto data zobrazia ako súčasť webovej aplikácie.

2.1.3 DOM based Cross Site Scripting

Ide o prípady, kedy sa ako súčasť webovej stránky vypíše hodnota elementu DOM danej stránky, ktorú má útočník pod kontrolou. To je možné ak sa napríklad do webovej stránky vypíše hodnota *document.referrer*, *alebo document.location* a ďalšie.

2.1.4 Cross Site Flashing

V tomto type zraniteľností sa manipuluje s ActionScript kódom, ktorý sa používa vo Flash aplikáciach.

2.1.5 Príklad

Tento príklad je prevzatý z [1]. Pri black-box testovaní na zraniteľnosť XSS útočník skúša rôzne testovacie vektory ktoré sa snažia obísť filtrovacie mechanizmy aplikácie. Predpokladajme, že súčasťou webovej aplikácie je nasledujúci kód:

```
<?
$re = "/<script[^ >]+src/i";
if (preg_match($re, $_GET['var'])) {
        echo "Filtered";
        return;
}
echo "Welcome " . $_GET['var'] . "!";
? >
```

V tomto prípade regulárny výraz filtruje vstupy typu:

```
<script [ľubovoľný znak okrem >] src
```

Tento filter je vhodný pre výrazy ako napríklad:

```
<script src="http://www.attacker.com/xss.js" > </script>
```

Potenciálny útočník ale tento filter dokáže obísť, ak do atribútu medzi *script* a *src* vloží znak ">", napríklad takto:

```
<script a=">" src="http://www.attacker.com/xss.js" > </script>
```

Táto problematika je podrobnejšie rozobraná napríklad v [5]

2.2 SQL injection

Táto zraniteľnosť sa nachádza spravidla v serverovej časti webovej aplikácie, kde sa vytvára dotaz do databáze na základe nesprávne skontrolovaného užívateľského vstupu. Za vhodných podmienok šikovný užívateľ dokáže zmanipulovať daný SQL dotaz. Útočník môže pomocou techniky SQL injection získať prístup k citlivým datam z databáze, modifikovať tieto data, vykonať administrátorské operácie na databáze (shutdown databázy), prípadne získať obsah súboru nachádzajúceho sa na filesystéme databázy. [1]

Podľa [1] rozlišujeme nasledujúce typy SQL injection

1. Inband - Data získavame z rovnakého komunikačného kanála, akým

sme injektovali SQL kód. Je to priamočiary spôsob útoku, kedy su získané data prezentované ako súčasť webovej aplikácie.

2. Out-of-band - Data získavame z iného kanála (Napríklad z výsledkov dotazu je generovaný email, ktorý je odoslaný útočníkovi)

3. Inferential - Nedochádza k doslovnému transferu citlivých dát - útočník získava informácie pozorovaním reakcií DB serveru na špeciálne zvolené dotazy

Nástroj *WebCop* podporuje detekciu *Inband SQL Injection*.

Nezávisle na hore prezentovaných typoch SQL injekcii, úspešný SQL injection útok vyžaduje vytvorenie syntakticky správneho SQL dotazu. V prípade že aplikácia vracia chybovú hlášku generovanú nesprávnym dotazom, je často možná konštrukcia syntakticky správneho dotazu na základe práve takýchto chybových hlášok. Ak aplikácia skríva detaily chybových hlášok, útočník musí vytvoriť úspešný útok pozorovaním správania sa webu na rôzne čiastkové jednoduchšie SQL injekcie. Tento prípad je známy aj pod pojmom *Blind SQL Injection*.

2.2.1 Príklad

V [1] je uvedený tento príklad: Predpokladajme, že webová aplikácia prevádza užívateľskú autentifikáciu nasledujúcim SQL dotazom:

```
SELECT * FROM Users WHERE
        Username="$username" AND
        Password="$password"
```

Ak tento dotaz vráti riadok z tabuľky *Users*, znamená to že v tejto tabuľke existuje užívateľ s príslušným heslom. V takom prípade je prístup do webovej aplikácie umožnený, v opačnom prípade sa prístup do webovej aplikácie odoprie. Login a heslo sa do webovej aplikácie zadáva zvyčajne pomocou prihlasovacieho formulára. útočník môže zadat nasledujúce údaje:

```
$username = 1" or "1" = "1
$password = 1" or "1" = "1
```

Ak tieto vstupné data nebudú ošetrené, vykoná sa nasledujúci SQL dotaz:

```
SELECT * FROM Users WHERE
        Username="1" or "1" = "1" AND
        Password="1" or "1" = "1"
```

Pretože tento dotaz vracia riadky z tabuľky *User*, webová aplikácia útočníkovi umožní sa prihlásiť bez znalosti hesla. V niektorých systémoch bude prvý vráteny riadok administrátorsky užívateľ, čo môže mať za následok prihlásenie do aplikácie s administrátorskými privilégiami.

2.3 Cross Site Request Forgery

Pri tomto útoku útočník spôsobí koncovému užívateľovi spustenie neželaných akcií v kontexte webovej aplikácie ku ktorej je užívateľ momentálne prihláseny. S menšiou dávkou social engineeringu (odoslanie odkazu/obrázku mailom, chatom) si koncový užívateľ vyvolá v browseri zobrazenie webovej stránky na adrese pripravenej útočníkom. To môže mať za následok odoslanie formulárov do zraniteľných skriptov webovej aplikácie, prípadne vyvolanie ďalších akcií v kontexte danej webovej aplikácie. Táto problematika je podrobnejšie rozobraná napríklad v [1].

2.3.1 Príklad

Predpokladajme, že máme webovú aplikáciu ktorá ponúka prístup k mailovej schránke. Súčasťou tejto aplikácie je aj formulár pomocou ktorého je možné definovať filtre na prichádzajúcu poštu a podľa týchto filtrov prichádzajúce maily preposielať na inú adresu, prípadne mazať. Ďalej predpokladáme, že tento formulár nie je chránený heslom, systémom *CAPTCHA* ani žiadnym skrytým náhodným tokenom ktorý je súčasťou formulára a ktorého pravosť sa pri spracovaní formulára kontroluje.

Útočník môže obeti zaslať mail s odkazom na svoju stránku, na ktorú umiestnil vyplnený formulár identický s tým, ktorý sa vo webovej aplikácii používa na definovanie filtrov prichádzajúcej pošty. Ak je parameter *id* tohto for-

mulára na útočníkovej webovej stránke rovný hodnote "fform", môže útočník element *body* na svojej stránke uviesť takto:

```
<body onload="document.getElementById('fform').submit()" >
```

To bude mať za následok automatické odoslanie formulára webovej aplikácii pri načítaní útočníkovej webovej stránky. V prípade že obeť je momentálne vo webovej aplikácii prihlásena, webový prehliadač spolu s formulárom na server odošle aj autentizačné cookies a útočník týmto spôsobom dokáže obeti nastaviť ľubovoľný filter na prichádzajúcu poštu. Samozrejme útočník má veľa možnosti ako tento útok zamaskovať aby si užívateľ nič nevšimol. Stačí že na svoju stránku umiestni napríklad miniatúrny *iframe* a formulár s javascript-ovým kódom na jeho automatické odoslanie bude súčasťou dokumentu zobrazeného v elemente *iframe*.

2.4 Remote File Inclusion

Táto zraniteľnosť sa môže nachádzať v serverovej časti webovej aplikácie (napr. php skript), kde sa inkluduje ďalší skript, ktorého názov (prípadne jeho časť) má pod kontrolou užívateľ. Útočník tam môže dosadiť adresu svojho skriptu, ktorý mu prevedie želané akcie. Jedná sa napríklad o predávanie názvu skriptu parametrom URL, ktorý nie je dostatočne kontrolovaný. Práve túto variantu dokáže nástroj *WebCop* detekovať. Táto téma je podrobnejšie spracovaná napríklad v [4].

Samotný kód, ktorý je *RFI* zraniteľný môže vyzerať podobne ako je uvedené v nasledujúcej ukážke na *LFI* zraniteľnosť. To či sa *Remote File Inclusion* dá skutočne previesť často závisí na ďalších faktoroch. V prípade php aplikácií sú relevantné predovšetkým nastavenia nasledujúcich príznakov: *register_globals*, *allow_url_fopen*, *allow_url_include*.

2.5 Local File Inclusion

Pri tejto zraniteľnosti sa útočník snaží do webovej aplikácie zakomponovať súbory nachádzajúce sa na danom serveri, ku ktorým by obyčajne užívateľ nemal mať prístup. Jedna sa napríklad o skripty na sťahovanie súborov, ktoré nekontrolujú dostatočne meno súboru na stianutie. Pomocou takýchto

skriptov môže útočník získať prístup na unixe napríklad k súboru /etc/passwd. V prípade že sa takýto súbor vo webovej aplikácii ešte prevedie interpretom (napríklad php), môže útočník priamo určovať (v tomto prípade php) príkazy, ktoré sa v rámci webovej aplikácie prevedú. Takáto situácia nastane ak napríklad užívateľ dokáže zariadiť, aby sa do aplikácie nainkludoval Apache log, do ktorého útočník predtým vhodnými webovými dotazmi vložil svoj (php) kód. Tejto téme sa podrobnejšie venuje napríklad [4].

2.5.1 Príklad

Nasleduje reálny príklad zraniteľnosti *LFI* v systéme *CodeDB* zverejnený v [6]. Zraniteľný kód vyzeral takto:

```
<?
$lang = htmlspecialchars($_GET['lang']);
if (file_exists('templates/' . $lang . '_middle.php'))
        include('templates/' . $lang . '_middle.php')
? >
```

Útočník mohol pomocou tohto kódu získať napríklad súbor */etc/passwd* takýmto spôsobom:

```
./wget http://[host]/[codeDB_path]/list.php?lang=../../../../etc/passwd%00
```

Na tomto príklade je predvedená aj technika *Null Byte Injection*, keď útočník do mena súboru vkladá byte s hodnotou nula, čím spôsobí ignorovanie prípony ktorá sa v kóde skriptu pridáva k názvu inkludovaného súboru.

2.6 Upload vulnerabilities

V tomto prípade sa jedná o zraniteľnosť, kedy webová aplikácia umožňuje upload súborov, ktorých prítomnosť na serveri môže útočník zneužiť. Typickým príkladom je aplikácia, ktorá umožní upload skriptov, ktoré server pri požiadaní o ich zobrazenie zinterpretuje. Veľmi zaujímavým príkladom zneužitia tejto zraniteľnosti je technika *GIFAR*, ktorá získala prvé miesto v súťaži *Top Ten Web Hacking Techniques of 2008*. Tejto téme sa venuje [8].

Kapitola 3

Analýza problému

3.1 Prehľad existujúcich riešení

3.1.1 w3af - Web Application Attack and Audit Framework

Autorom nástroja *w3af* [9] je Andres Riancho, zakladateľ a information security researcher v spoločnosti Bonsai. Cieľom tohto Open Source projektu je vytvorenie rozšíriteľného frameworku na vyhľadávanie a zneužívanie zraniteľností webových aplikácii. Tento projekt je implementovaný v skriptovaciom jazyku Python. Celý tento framework je rozdelený do dvoch častí, core a pluginy. Core koordinuje procesy a poskytuje služby ktoré využívajú jednotlivé pluginy. Pluginy poskytujú samotnú funkcionalitu ako prehľadávanie štruktúri webovej aplikácie a samotné black-box testovanie.

Z užívateľského hľadiska je tento nástroj príjemný. Okrem prehľadného konzolového prostredia poskytuje aj grafické užívateľské rozhranie. Pri testovaní tohto nástroja sa mi ale často stávalo, že test po čase spadol.

Ako základnú nevýhodu tohto nástroja vidím predovšetkým to, že samotný užívateľ nemá možnosť definovať aké attack vektory sa pri konkrétnych testoch použijú.

3.1.2 Wapiti

Nástroj *Wapiti* [10] je Open Source skener zraniteľnosti webových aplikácií. Jeho autorom je Nicolas Surribas. Podobne ako predchádzajúci framework, je tento nástroj implementovaný v skriptovacoim jazyku Python. Spoločnou črtou frameworku *w3af* a nástroja *Wapity* je, že ani jeden z týchto nástrojov nezávisia na externej databáze zraniteľností. Oba nástroje sa snažia vybrané typy webových zraniteľností samostatne vyhľadávať technikou black-box testing.

Tento nástroj sa snaží byť užívateľský príjemný. Dojem z neho však kazí jeho nestabilita, kde aj keď som testoval najnovšiu verziu tohto nástroja, stále mi počas testu na vybraných webových aplikáciach skôr alebo neskôr spadol.

Podobne ako to je s nástrojom *w3af*, aj tu vnímam ako jeho základnú nevýhodu prakticky nulovú možnosť užívateľa špecifikovať aké data sa pri black-box testovaní budú na webovú aplikáciu odosielať.

3.1.3 Nikto

Nástroj *Nikto* [11] je Open Source skener webových serverov. Jeho autorom je Chris Sullo, CFO spoločnosti *CIRT, Inc.* Momentálne dokáže prevádzať testy na prítomnosť asi 3500 potenciálne nebezpečných súborov/CGI skriptov, verzií asi 900 webových serverov a konkrétnych version-specific problémov na viac ako 250 webových serverov.

Z toho že nástroj *Nikto* je primárne skener webových serverov a nie skener webových aplikácií vyplýva ďalší principiálny rozdiel ktorým sa odlišuje od predchádzajúcich riešení. Tento nástroj prevádza testy na konkrétne problémy ktoré má uvedené vo svojej databáze, tým je účinnosť nástroja obmedzená rozsiahlosťou a frekvenciou updatovania danej databázy. Tento spôsob testovania sa označuje aj ako *Signature Based Testing*.

3.1.4 Nessus

Autorom a správcom projektu *Nessus* [12] je spoločnosť *Tenable Network Security, Inc.* Nessus je skener zameraný na audit bezpečnosti počítačových sietí. V rámci tejto funkcionality ponúka aj možnosti na testovanie bezpečnosti

webových serverov a webových aplikácií.

Podobne ako nástroj *Nikto* aj tento nástroj je závislý na databáze známych konkrétnych bezpečnostných chýb, ktorých prítomnosť testuje. Okrem toho však tento nástroj umožňuje prehľadávanie štruktúry dokumentov webovej stránky a samostatné vyhľadávanie vybraných typov bezpečnostných zraniteľností. Aj v tomto nástroji však chýba možnosť konfigurácie konkrétnych attack vektorov pomocou ktorých sa prevádza následné black-box testovanie webovej aplikácie.

3.2 Požiadavky na nástroj WebCop

Základným požiadavkom, ktorý kladiem na nástroj *WebCop* je principiálna schopnosť detekovať typy webových zraniteľností ktoré sú popísané v kapitole 2. To či nástroj v konkrétnej webovej aplikácii odhalí konkrétnu chybu bude určené predovšetkým konfiguráciou odpovedajúceho testu ktorá určí aké data sa na server odošlu a za akých podmienok sa prítomnosť zraniteľnosťi vyhodnotí pozitívne.

Nástroj *WebCop* nebude závislý na žiadnej externej databáze zverejnených bezpečnostných chýb v konkrétnych webových aplikáciach. Tento nástroj bude tieto chyby aktívne vyhľadávať systémom black-box testing.

Konfigurácia a výstup nástroja bude vo formáte XML. Konfigurovateľný bude nie len nástroj ako celok, ale užívateľ bude môcť konfigurovať aj jednotlivé moduly zodpovedné za prevedenie konkrétnych testov. A to na takej úrovni, že užívateľ bude mať priamu kontrolu nad datami ktoré sa odošlú na server. Takisto špecifikácia, za akých podmienok sa prítomnosť zraniteľnosťi vyhodnotí pozitívne je ponechaná v čo najväčšiej miere na užívateľovi.

Nástroj bude poskytovať jednotné programátrorské rozhranie na definovanie nových testov bezpečnostných zraniteľností. Konkrétne pridanie ďalšieho modulu do nástroja na testovanie novej zraniteľnosti bude pre programátora predstavovať vytvorenie triedy ktorá dedí z na to určenej bázovej triedy. Táto nová trieda bude implementovať potrebné rozhranie.

Nástroj bude napísaný v jazyku C++, je určený pre platformu Unix. Nástroj bude pripravený pre beh na platformách FreeBSD, GNU/Linux.

Kapitola 4

Programátorská dokumentácia

4.1 Použité algoritmy testovania zraniteľností

4.1.1 Cross Site Scripting

Pri testovaní zraniteľností *XSS* užívateľ nakonfiguruje pravidlá, podľa ktorých má nástroj vypĺňať webové formuláre. Nástroj na každý formulár, ktorý nájde na stránkach danej webovej aplikácie, aplikuje tieto pravidlá. V dokumente, ktorý sa mu vráti ako odpoveď na odoslanie formulára skontroluje prítomnosť užívateľom definovanej XSS injekcie. Ak je prítomná a nachádza sa v spustiteľnej podobe (nie je zakomentovaná ani chránená uvodzovkami) test vyhodnotí pozitívne.

4.1.2 SQL Injection

Pri testovaní zraniteľností *SQL Injection*, podobne ako pri *XSS* užívateľ nakonfiguruje pravidlá, podľa ktorých má nástroj vypĺňať webové formuláre. Nástroj postupne odošle všetky formuláre webovej aplikácie vyplnené podľa týchto pravidiel. Po každom odoslaní formulára sa testuje prítomnosť identifikačného reťazca v odpovedi webovej aplikácie. Rozdiel oproti testovaniu zraniteľností XSS spočíva v tom, že sa nekontroluje, či sa špecifikovaný reťazec nachádza v HTML dokumente v spustiteľnej podobe.

4.1.3 Cross Site Request Forgery

Použitie tohto nástroja na automatické vyhodnocovanie webových formulárov z hľadiska CSRF zraniteľnosti je možné, iba ak má daná webová aplikácia časť chránenú užívateľskou autentifikáciou. Táto autentifikácia musí prebiehať pomocou HTML formulára na login a heslo. Po úspešnom prihlásení ďalej nástroj predpokladá, že od webového servera obdrží cookies a s pomocou týchto cookies bude nástroju umožnená navigácia v heslom chránenej časti webovej aplikácii. Nástroj po prihlásení začne túto webovú aplikáciu prehľadávať a pre každý webový formulár, ktorý nájde, skontroluje či je dostupný aj pre neprihláseného užívateľa (Bez autentifikačných cookies). V prípade že formulár je nedostupný pre neprihláseného užívateľa, nástroj skontroluje či je súčasťou formulára randomizačný anti-CSRF token. V prípade že nie, test daného formulára na prítomnosť CSRF zraniteľnosti sa vyhodnotí pozitívne. Ak je súčasťou skúmaného webového formulára HTML element na vloženie hesla, nástroj automaticky tento formulár vyhodnotí negatívne.

Tento spôsob testovania CSRF zraniteľnosti je nastavený na situáciu bežných webových aplikácii dnes dostupných na internete. Samotný test je v tomto prípade bezpečný v zmysle, že nedochádza k samotnému odosielaniu webových formulárov. Tento test je založený na predpoklade, že potenciálne nebezpečný formulár je iba formulár, ku ktorému ak chce užívateľ získať prístup, musí sa najprv prihlásiť do webovej aplikácie. Týmto eliminujem rôzne vyhľadávacie a im podobné formuláre vo webových aplikáciach, ktoré síce môžu byť CSRF zraniteľné, ale pri nich útok nemá praktický význam. Ďalej je test založený na predpoklade, že jediná účinná ochrana webového formulára pred CSRF útokom je ochrana heslom, alebo náhodným tokenom.

Som si vedomý, že existujú aj iné účinné ochrany webových formulárov pred zraniteľnosťami CSRF. Jedná sa napríklad o použitie systému CAPTCHA, prípadne použitie ďalších kódov zaslaných iným komunikačným kanálom (Verifikácia SMS kódom). Tento algoritmus testovania, ktorý som tu popísal sa mi zdal ako rozumný kompromis medzi tým čo sa dá relatívne jednoducho naimplementovať a početnosťou výskytu tzv. false positives s ohľadom na situáciu dnes bežných webových aplikácii.

4.1.4 Remote File Inclusion

Pri definovaní testov tejto zraniteľnosti užívateľ nadefinuje HTTP get parametre, pomocou ktorých chce skúšať inkludovanie svojho skriptu do webovej aplikácie. Ďalej zadá adresu svojho skriptu a kontrolný reťazec, pomocou ktorého bude nástroj zisťovať či sa skript skutočne zinterpretoval.

Kedykoľvek nástroj pri prehľadávaní štruktúri webovej aplikácie nájde URL adresu v ktorej je použitý aspoň jeden z definovaných GET parametrov, dosadí za jeho hodnotu adresu definovaného skriptu. V odpovedi na výsledný HTTP get dotaz skontroluje, či sa tam nachádza určený reťazec.

4.1.5 Local File Inclusion

Testovanie zraniteľností *LFI* sa prevádza rovnakým spôsobom ako testovanie zraniteľností *RFI*. Rozdiel je v tom že užívateľ pri konfigurácii testu zraniteľnosti LFI má možnosť ako hodnotu parametru ktorá sa má dosadiť za potenciálne nebezpečné GET parametre uviesť znak @. Tento znak bude zastupovať meno skriptu, ktorý sa práve testuje.

4.1.6 File Upload Vulnerabilities

Pri konfigurovaní testu na tento typ zraniteľnosti užívateľ definuje súbor, ktorý sa ma nástroj pokúsiť uploadnuť kedykoľvek nájde vo webovej aplikácii formulár ktorý to umožňuje. Ďalej užívateľ určí URL adresy, kde by na serveri mohol uploadnutý súbor byť k dispozícii. Pri konfigurácií takisto užívateľ definuje unikátny reťazec ktorý je súčasťou uploadovaného súboru. Potom čo nástroj uploadne určený súbor, vyskúša HTTP get požiadavky na špecifikované URL adresy. Ak ako odpoveď na takýto požiadavok dostane dokument obsahujúci určený reťazec, nástroj vyhodnotí tento test pozitívne.

4.2 Softwarová architektúra

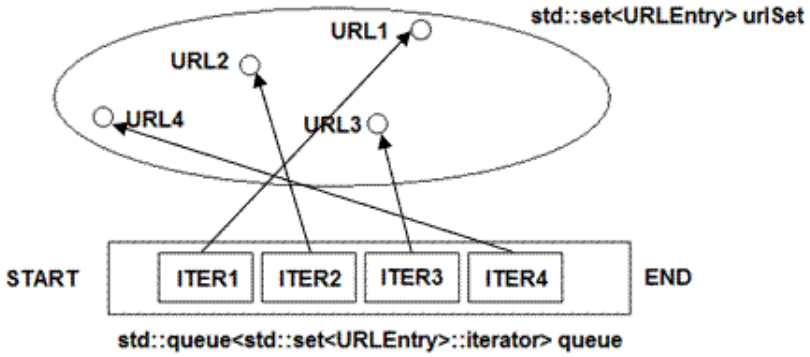
Nástroj obsahuje vlastnú implementáciu potrebnej podmnožiny protokolu HTTP 1.1. Táto komunikácia je zapúzdrená v triede *HTTPClient*. K parsovaniu XML konfigurácie je použitá knižnica *expat*. K sieťovej komunikácii

na úrovni protokolu TCP využíva tento nástroj systémové volania UNIXu. Nástroj je viacvláknový, k správe vlákien používa *pthreads* API.

Celý proces testovania webovej aplikácie zapúzdruje trieda *TestingSystem* deklarovaná v súbore *TestingSystem.h*. Táto trieda si drží v STL kontajneri *std::vector* referencie na objekty typu *AbstractTester*, čo je abstraktná trieda z ktorej dedia objekty reprezentujúce jednotlivé testy. Trieda *TestingSystem* obsahuje aj referenciu na objekt typu *WebCrawler*, čo je trieda zapúzdrujúca proces prehľadávania štruktúry webovej aplikácie.

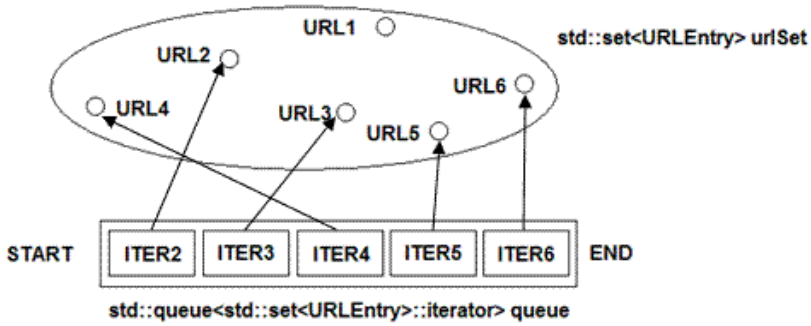
Po štarte programu spustí *TestingSystem* na triede *WebCrawler* preihľadávanie webovej aplikácie podľa parametrov konfigurácie programu. Až tento proces skončí, začne trieda *TestingSystem* podľa povahy nakonfigurovaných testov spúšťať jednotlivé testy. Na konci programu trieda *TestingSystem* vypíše do výstupného súboru zoznam nájdených zraniteľností.

Trieda *WebCrawler* prehľadáva štruktúru webovej aplikácie systémom *Breadth First Search*. Nájdené URL adresy aplikácie si uchováva v *STL* kontajneri *set*. Tento šablonovaný kontajner je parametrizovaný typom *URLEntry* ktorý reprezentuje jednotlivé URL adresy. Táto trieda obsahuje o URL adrese ďalšie informácie, ako napríklad naparsované GET parametre, či URL kde sa táto adresa vo webovej aplikácii našla. *WebCrawler* si v *STL* kontajneri *queue* udržuje frontu iterátorov na objekty *UrlEntry* nachádzajúce sa v spomínanom množinovom kontajneri *set*. Vždy keď sa nájde nová URL adresa, vloží sa do spomínanej množiny. Iterátor na tento novo vzniknutý objeku *URLEntry* sa vloží do spomínanej fronty. Samotné prehľadávanie do šírky sa realizuje postupným vyberaním iterátorov na URL adresy zo začiatku fronty a následným ukladaním iterátorov na nové adresy na koniec fronty. Tento proces je znázornený na nasledujúcich obrázkoch.



Obr. 4.1: *WebCrawler* obsahuje 4 URL adresy.

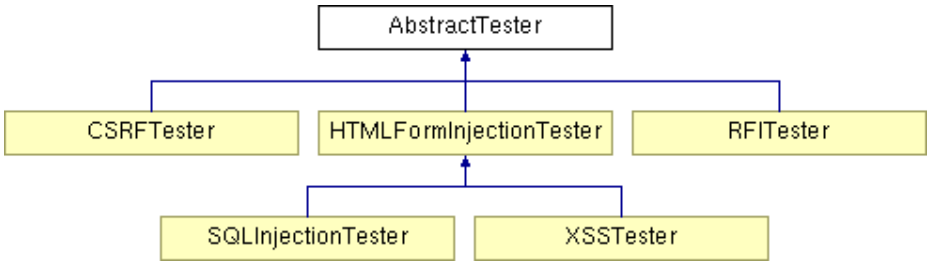
Na tomto obrázku je zobrazený stav, keď *WebCrawler* pri prehľadávaní štruktúri webovej aplikácie našiel štyri URL adresy. Na každú z týchto adries má vo fronte iterátor.



Obr. 4.2: *WebCrawler* pri prehľadávaní do šírky o krok ďalej.

Pri ďalšiom prehľadávaní sa z fronty vybral prvý prvok a prehľadal sa HTML dokument na odpovedajúcej URL adrese. Na ňom sa našli odkazy na ďalšie dve URL adresy (URL5 a URL6). Tieto nové adresy sa pridali do množiny adries a iterátory k týmto novým prvkom sa pridali na koniec fronty.

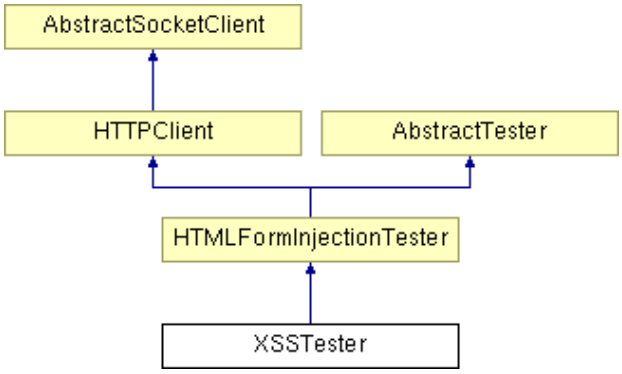
Jednotlivé testy na zraniteľnosti webových aplikácií su reprezentované samostatnými triedami. (Výnimkou sú testy na zraniteľnosti RFI a LFI, ktoré sú reprezentované jednou triedou *RFITester*) Tieto triedy sú usporiadané v nasledujúcej hierarchii:



Obr. 4.3: Triedy reprezentujúce webové zraniteľnosti.

Takéto usporiadanie umožnuje triede *TestingSystem* manipulovať s objektami reprezentujúcimi jednotlivé webové zraniteľnosti polymorfným spôsobom.

Okrem toho, že tieto objekty sú potomkami triedy *AbstractTester*, dedia príslušné triedy aj z triedy *HTTPClient*. To týmto objektom pridáva potrebnú funkcionalitu na prevádzanie testov webových zraniteľností. Konkrétne začlenenie napríklad objektu *XSSTester* do hierarchie tried vyzerá takto:

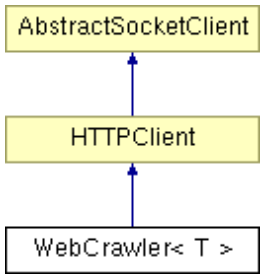


Obr. 4.4: Trieda XSSTester.

Nástroj na viacerých miestach kódu využíva callback funkcie. Napríklad takýmto spôsobom trieda *WebCrawler* signalizuje koniec procesu prehľadávania

webovej aplikácie. Podobne objekty reprezentujúce jednotlivé testy signalizujú callback funkciou koniec testu. Tento pojem callback funkcie je zapúzdrený v šablone *CallBackFunction*, ktorá je parametrizovaná triedou, ktorej member funkciu tento callback reprezentuje.

Podobne vyššie spomínaná trieda *WebCrawler* je v skutočnosti implementovaná ako šablona parametrizovaná triedou, ktorej member funkciu *WebCrawler* zavolá po skončení prehľadávania webovej aplikácie. V kóde je táto šablona parametrizovaná jednak triedou *TestingSystem* - kedy *WebCrawler* informuje *TestingSystem* o skončení prehľadávania, jednak triedou *CSRFTester*, ktorý používa vlastný *WebCrawler* na prehľadanie časti webovej aplikácie chránenej autentifikáciou. Začlenenie šablony *WebCrawler* do hierarchie tried vyzerá nasledovne:



Obr. 4.5: Šablona WebCrawler.

Ako je zrejmé z predchádzajúcich obrázkov, trieda *HTTPClient* je potomkom triedy *AbstractSocketClient*. Trieda *AbstractSocketClient* zapúzdruje riadenie TCP spojení. Umožňuje vytváranie nových spojení (vrátane špecifikovania connection timeoutu), ich rušenie, odosielanie dát a pri výskyte určitých udalostí (príchod dát zo serveru) vyvolá príslušne obslužné virtuálne funkcie, ktoré implementujú klienti tejto triedy.

Kapitola 5

Užívateľská dokumentácia

5.1 Základné informácie

5.1.1 Softwarové požiadavky

Nástroj *WebCop* je naimplementovaný pre platformu *UNIX*, využíva knižnicu jazyka C *expat* na prácu s XML datami. Pre úspešné skompilovanie a riadne fungovanie má táto aplikácia nasledujúce požiadavky:

1. Platforma UNIX
2. Prekladač jazyka C++
3. GNU make
4. expat XML library
5. POSIX threads
6. Doxygen (Pre programátorskú dokumentáciu)

5.1.2 Adresárová štruktúra

Súbory, ktoré sú súčasťou nástroja sú usporiadané podľa nasledujúcej hierarchie:

- Hlavný adresár
  - *build* - Adresár do ktorého sa kompilujú zdrojové súbory.

- *doc* - Obsahuje dokumentáciu
  - *user* - Užívateľská dokumentácia
  - *programmer* - Programátorská dokumentácia
- *examples* - Príklady konfiguračných súborov
  - *config* - Konfiguračné súbory config
  - *data* - datový súbor, využía sa pri testovaní upload zraniteľností.
  - *input* - Konfiguračné súbory input.txt
  - *output* - Príklady výstupov
- *src* - Obsahuje zdrojové kódy

5.2 Kompilácia

Zdrojové kódy sú súčasťou CD, prípadne je možné ich stiahnuť z SVN archívu na adrese

`https://webcop.svn.sourceforge.net/svnroot/webcop`.

V základnom adresári sa nachádza skript *configure*. Je vygenerovaný pomocou nástroja *GNU autoconf*. Príkazom ./configure sa z predlôh Makefile.in vygenerujú súbory makefile potrebné pre úspešné skompilovanie zdrojových kódov. Do príslušného makefile súboru sa pritom doplní cesta ku knižnici *expat*. (Štandardne */usr*, prípadne */usr/local*). V prípade, že knižnica *expat* nebola nainštalovaná do defaultného adresára, je potrebné túto cestu uviesť v parametri –with-expat skriptu *configure*.

Príklad: knižnica expat bola nainštalovaná do adresára /usr/home/pepo/expat. Potrebné súbory makefile vygenerujeme príkazom

./configure –with-expat=/usr/home/pepo/expat.

Príkazom ./configure sa vygenerovali v základnom adresári okrem súboru *Makefile* aj súbory *config.status* a *config.log*. *Config.status* umožňuje znovu vygenerovať príslušné makefile súbory, súbor *config.log* obsahuje predovšetým debugovacie informácie.

Následne príkazom make skompilujeme zdrojové kódy do spustiteľného programu *tester*. Je nutné použiť *GNU make*. Na *GNU/linuxe* je to príkaz *make*. na *FreeBSD*, *OpenBSD* je to príkaz *gmake*.

5.2.1 Makefile - ciele

Súbor Makefile nachádzajúci sa v základnom adresári má pripravené nasledujúce ciele:

- *all* - Defaultný cieľ. Skompiluje zdrojové kódy do spustiteľného programu.
- *clean* - Zmaže skompilované súbory.
- *debug* - Skompiluje zdrojové kódy do spustiteľného programu. Ten bude obsahovať aj debugovacie informácie.
- *depend* - Do makefilu vygeneruje závislosti zdrojových súborov spôsobené direktívami preprocesora #include. (Pre účely programátora)
- *distclean* - Zmaže skompilované súbory a súbory vygenerované skriptom *configure*. Zmaže aj programátorskú dokumentáciu.
- *doc* - Vygeneruje programátorskú dokumentáciu. (Vyžaduje *doxygen*)
- *docclean* - Zmaže programátorskú dokumentáciu.

5.3 Konfigurácia - beh programu

Nástroj *WebCop* potrebuje k spustreniu dve konfiguračné súbory. Musia sa nachádzať v adresári so spustiteľným súborom. Ide o súbory *config* a *input.txt*.

Súbor config obsahuje základné informácie popisujúce kde sa majú webové zraniteľnosti hľadať. Tento súbor obsahuje riadky typu PARAM=VALUE, kde PARAM je názov konfigurovaného parametru a VALUE je jeho hodnota. Riadky začínajúce sa # sa ignorujú. Nasleduje výpis konfiguračných parametrov.

- *domain* - Internetová doména, kde sa nachádza testovaná webová aplikácia.
- *url* - Počiatočná URL adresa webovej aplikácie. Prehľadávanie domény začne z tejto adresy.
- *output* - Meno výstupného súboru. Bude obsahovať popis nájdených webových zraniteľností.
- *depth* - Limit na hĺbku BFS prehľadávania webových stránok aplikácie. Ak je nastavený na záporné číslo, doména sa prehľadá celá.
- *crawlStop* - Pauza v sekundách pred prehľadávaním ďalšiej webovej stránky
- *workStop* - Pauza v sekundách pred testovaním ďalšiej zraniteľnosti.

5.4 Konfigurácia - testovanie zraniteľností.

Konfigurácia testovaných zraniteľností je vo formáte XML v súbore *input.txt*. Ak hodnota nejakého atribútu má obsahovať špeciálny znak, musí sa použiť odpovedajúca XML entita. Každá zraniteľnosť je definovaná pomocou párového tagu *vulnerability*:

```
<vulnerability type="typ" id="id"> </vulnerability>
```

Parameter *typ* určuje typ testovanej zraniteľnosti, parameter *id* definuje identifikátor definovaného testu.

5.4.1 XSS

Testy zraniteľností XSS sú definované podľa nasledujúcej schémy:

```
<vulneriability type="xss" id="ID">
        Tu sa nachádzajú pravidlá, ako má nástroj vypĺňať webové
        formuláre. Každé pravidlo ide na samostatný riadok.
        <success>
                Tu sa nachádzajú podmienky. Ak je splnená aspoň jedna,
                výsledok sa vyhodnotí ako XSS zraniteľnosť.
                Každá podmienka ide na samostatný riadok.
        </success>
</vulnerability>
```

Pravidlá na vypĺňanie webových formulárov možu byť špecifické alebo všeobecné. Všeobecné pravidlo znamená, že všetkým prvkom daného webového formulára sa nastaví hodnota value na určenú hodnotu. Špecifické pravidlo určuje hodnotu atribútu value iba pre vybraný element webového formulára. Špecifické pravidlo má vyššiu prioritu ako všeobecné pravidlo. Všeobecné pravidlo má formát:

```
<input type="any" value="hodnota" />
```

Špecifické pravidlo má formát:

```
<input type="element" value="hodnota" />
```

Kde *element* značí vybraný HTML element webového formulára. V prípade, že ako hodnota je uvedený znak *, ponechá sa danému prvku pôvodná hodnota.

Podmienky určujú textový reťazec, ktorý sa musí nachádzať na webovej stránke zobrazenej ako výsledok odoslaného formulára. Tento reťazec sa naviac musí nachádzať v spustiteľnej podobe. (Nie je zakomentovaný a nie je chránený uvodzovkami) Podmienka má formát:

```
<contains value="hodnota" />
```

Nasleduje kompletný príklad konfigurácie testu na XSS zraniteľnosť.

```
<vulneriability type="xss" id="1">
        <input type="text" value="&lt;script&gt;alert(34)&lt;/script&gt;"/>
        <input type="hidden" value="*" />
        <success>
                <contains value="&lt;script&gt;alert(34)&lt;/script&gt;" />
        </success>
</vulnerability>
```

5.4.2 SQL injection

Definícia testu na SQL injekciu je obdobná testu na XSS zraniteľnosť:

```
<vulneriability type="sql" id="ID">
        Tu sa nachádzajú pravidlá, ako má nástroj vypĺňať webové
        formuláre. Každé pravidlo ide na samostatný riadok.
        <success>
                Tu sa nachádzajú podmienky. Ak je splnená aspoň jedna,
                výsledok sa vyhodnotí ako XSS zraniteľnosť.
                Každá podmienka ide na samostatný riadok.
        </success>
</vulnerability>
```

Pravidlá na vypĺňanie webových formulárov sú rovnaké ako pri zraniteľnosti XSS.

Podmienky určujú textový reťazec, ktorý sa musí nachádzať na webovej stránke zobrazenej ako výsledok odoslaného formulára. Podmienka má rovnaký formát ako v prípade XSS zraniteľnosti. Už sa ale nekontroluje či sa určený textový reťazec na webovej stránke nachádza v spustiteľnej podobe.

5.4.3 Cross Site Request Forgery

Definícia testu na CSRF sa riadi nasledujúcou schémou:

```
<vulneriability type="csrf" id="ID">
        <csrf login="tester" pwd="secretPassword"
                loginurl="http://profinzer.zaridi.to" />
        <skip>
                Tu sa nachádzajú vzroy URL adries,
                ktoré sa pri prehľadávaní vynechajú.
        </skip>
</vulnerability>
```

Login *tester* a heslo *secretPassword* sa nahradia platnými prihlasovacími údajmi. Takisto ako parameter loginurl sa uvedie príslušná adresa HTML dokumentu s prihlasovacím formulárom.

Aby pri prehľadávaní časti webovej aplikácie ktorá je chránená heslom nedošlo k neželaným akciám, je nutné uviesť vzory URL adries, ktoré sa z prehľadávania vylúčia. Vzory URL adries sa zadávajú vo formáte:

```
<url pattern="hodnota" />
```

Nasleduje príklad konfigurácie testu na CSRF zraniteľnosti:

```
<vulneriability type="csrf" id="3">
        <csrf login="tester" pwd="secretPassword"
                loginurl="http://profinzer.zaridi.to" />
        <skip>
                <url pattern="logout" />
                <url pattern="logoff" />
                <url pattern="unregister" />
        </skip>
</vulnerability>
```

5.4.4 Remote File Inclusion

Definícia testu na RFI zraniteľnosť vyzerá nasledovne:

```
<vulneriability type="rfi" id="ID">
        <getparam name="param_names" value="url" />
        <success>
                <contains value="unique_id" />
        </success>
</vulnerability>
```

Pričom *param_names* obsahuje zoznam (oddelené čiarkou) potenciálne nebezbečných get parametrov. *url* je adresa nášho skriptu, ktorý vypíše unikátny reťazec *unique_id*.

5.4.5 Local File Inclusion

Či sa podarilo stiahnuť zdrojový (php) súbor aplikácie sa dá testovať pomocou prítomnosti špeciálnych symbolov konkrétneho jazyka. (Napríklad pre php je to <?) Test na formuláre určené pre sťahovanie súborov zo serveru môže vyzerať napríklad takto:

```
<vulneriability type="lfi" id="4">
        <getparam name="file,fileName" value="@" />
        <success>
                <contains value="&lt;?" />
        </success>
</vulnerability>
```

5.4.6 Upload vulnerabilities

Konfigurácia testov na testovanie skriptov určených pre html upload sa riadi touto schémou:

```
<vulneriability type="upload" id="ID">
        Tu sa definujú informácie o lokálnom súbore, ktorý uploadujeme.
        <success>
                Tu sa uvádza zoznam URL adries, na ktorých
                by uploadnuty súbor mohol byť k dispozícii.
                Každá adresa ide na samostatný riadok.

                Ďalej sa tu definuje textový reťazec, ktorý
                sa na danej webovej stránke musí nachádzať.
        </success>
</vulnerability>
```

O lokálnom súbore, ktorý chceme uploadovať špecifikujeme jeho meno a jeho mime[1] typ. Tieto informácie zadávame vo formáte kompatibilným s pravidlami na vypĺňanie webových formulárov pri definovaní testov zraniteľností ako XSS, SQL injection. Presný formát je takýto:

```
<input type="file" value="fileName" mime="mimeType" />
```

URL adresy v tagu <success> sa definujú nasledovne:

```
<url value="adresa" />
```

Kde *adresa* je daná url adresa. Textový reťazec definujeme nasledovne:

```
<contains value="uniqueString" />
```

Kde *uniqueString* je špecifikovaný unikátny reťazec. Kompletná konfigurácia testu na zraniteľnosť upload skriptu môže vyzerať nasledovne:

[1] Multipurpose Internet Mail Extensions

```
<vulneriability type="upload" id="5">
        <input type="file" value="examples/data/hello.php"
                mime="text/plain" />"
        <success>
                <url value="upload/hello.php" />
                <url value="Upload/hello.php" />
                <url value="uploads/hello.php" />
                <url value="Uploads/hello.php" />

                < contains value="9876543210" />
        </success>
</vulnerability>
```

5.5 Výstup

Výstupný súbor tohto nástroja je určený v konfiguračnom súbore *config*. Pre každú zaznamenanú zraniteľnosť obsahuje výstupný súbor XML element vo formáte:

```
<vulnerability type="type"
        url="url"
        id="id"
        formname="formname"
        formid="formid"
        formaction="formaction" />
```

Význam jednotlivých parametrov elementu *vulnerability* je nasledovný:

- *url* - Url adresa na ktorej sa zraniteľnosť nachádza.
- *id* - Id zraniteľnosti. Podľa neho je možné spárovať zraniteľnosť nájdenú na webe s popisom zraniteľnosti v súbore input.txt
- *formname* - Hodnota atribútu name zraniteľného HTML formulára.

- *formid* - Hodnota atribútu id zraniteľného HTML formulára.
- *formaction* - Hodnota atribútu action zraniteľného HTML formulára.

5.6 Ukážka behu programu

S nástrojom sú distribuované predvyplnené konfiguračné súbory, ktoré sú nastavené tak že po spustení programu sa začne testovať modelová aplikácia na adrese `http://profinzer.zaridi.to/`. Testovanie tejto aplikácie teda spustíme príkazom:

```
./tester
```

Po spustení programu sa začne prevádzať samotný test. V závislosti na veľkosti webovej aplikácie môže trvať rôzne dlho. Celý program je možné kedykoľvek ukončiť stlačením klávesy enter. Po skončení testovania program na štandardný výstup vypíše hlášku *Press enter to exit*. Po jeho stlačení sa na štandardný výstup vypíše zoznam skenovaných URL adries a program skončí.

Na jednoduchšiom príklade ukážem prácu s týmto programom, od konfigurácie cez beh programu až po vyhodnotenie výsledkov. Na tomto demonštračnom príklade sa zameriam iba na chyby *RFI* a *SQL injection*. Konfiguračný súbor *config* vyzerá takto:

```
domain=profinzer.zaridi.to
url=http://profinzer.zaridi.to
output=output.txt
depth=-1
crawlStop=1
workStop=1
```

Táto konfigurácia špecifikuje, že testovať sa budú iba skripty vrámci domény *profinzer.zaridi.to*. Ďalej je uvedená úvodná stránka testovanej webovej aplikácie. Nasleduje meno súboru ktorý bude obsahovať záverečnú správu o nájdených zraniteľnostiach. Pretože parameter *depth* má hodnotu záporného čísla, aplikácia sa bude prehľadávať celá, hĺbka prehľadávania nebude obmedzená. Parameter *crawlStop* určuje, že počas prehľadávania webovej ap-

likácie sa pred skokom na nasledujúci dokument vo fronte počká 1 sekundu. Podobne parameter *workStop* určuje pauzu v sekundách medzi testovaním jednotlivých zraniteľnosti.

Konfiguračný súbor *input.txt* obsahuje definície testov na zraniteľnosti *RFI* a *SQL injection*. V *RFI* teste je špecifikovaná adresa `http://rocnikac.100webspace.net/index.txt`, kde sa nachádza php skript ktorý vypíše obsah súboru *index.php*. Okrem toho vypíše aj reťazec "index.php listing – start".

```
<vulnerability type="sql" id="1">
        <input type="any" value="pepo&quot; order" />
        <input type="hidden" value="*" />
        <success>
            <contains value="Warning" />
            <contains value="MySQL result" />
        </success>
</vulnerability>

<vulnerability type="rfi" id="ID">
        <getparam name="file,myurl,page"
            value="http://rocnikac.100webspace.net/index.txt" />
        <success>
            <contains value="index.php listing – start" />
        </success>
</vulnerability>
```

Počas behu program vypisuje na štandardný výstup rôzne informácie. Nasleduje ukážka výstupu programu počas fázy prehľadávania webovej aplikácie:

```
 ========================================
Number of URLs found so far: 1
Next url: http://profinzer.zaridi.to
Counter: 0
Queue size: 0
========================================

Sleeping for 1 seconds.
Connection terminated by remote server
profinzer.zaridi.to
Searching http://profinzer.zaridi.to
```

Ďalej prikladám ukážku výstupu programu počas samotného testovania:

```
Trying to get http://profinzer.zaridi.to
Connection terminated by remote server
state: loading_forms
url: http://profinzer.zaridi.to
Filling xss forms 0
Connection terminated by remote server
state: checking_forms
url: http://profinzer.zaridi.to
Checking for SQL injection on url http://profinzer.zaridi.to
SQL injection detected !!!
```

Po skončení testu sa správa o nájdených zraniteľnostiach nachádza v súbore *output.txt*. Tento súbor vyzerá nasledovne:

```
<vulnerability type="sql" url="http://profinzer.zaridi.to" id="1"
        formname=""
        formaction="http://profinzer.zaridi.to/login.php"
        formid="hidden" />

<vulnerability type="sql" url="http://profinzer.zaridi.to" id="1"
        formname=""
        formaction="http://profinzer.zaridi.to/index.php" formid="" />

<vulnerability type="sql" url="http://profinzer.zaridi.to/index.php" id="1"
        formname=""
        formaction="http://profinzer.zaridi.to/login.php" formid="row" />

<vulnerability type="sql" url="http://profinzer.zaridi.to/index.php" id="1"
        formname=""
        formaction="http://profinzer.zaridi.to/index.php" formid="" />

<vulnerability type="rfi" url="http://profinzer.zaridi.to/vuln2.php"
        id="2" />
```

Z tohto výstupu sa dá usúdiť, že nástroj *WebCop* našiel v našej webovej aplikácii jednu *RFI* zraniteľnosť a 4 *SQL Injection* zraniteľnosti. Pri pozornejšiom prezretí výstupu je jasné, že v skutočnosti boli odhalené iba dve zraniteľnosti *SQL injection*. Na každú z týchto zraniteľností nástroj natrafil dvakrát, totiž z dvoch rôznych URL adries.

Kapitola 6

Porovnanie s existujúcimi riešeniami

6.1 w3af - Web Application Attack and Audit Framework

Ako prvú výhodu nástroja *w3af* oproti nástroju *WebCop* vidím jeho platformovú nezávislosť, keďže je napísaný v Pythone. Nástroj *WebCop* je určený pre platformu unix a je napísany v jazyku C++. Na druhej strane je inštalačný proces nástroja *WebCop* jednoduchší, pretože nevyžaduje inštaláciu takého množstva knižníc tretích stran.

Dalšiou výhodou nástroja *w3af* je väčšie množstvo typov webových zraniteľností, ktoré pokrýva. Nástroj *WebCop* momentálne pokrýva zraniteľnosti *XSS*, *SQL injection*, *RFI*, *LFI*, *CSRF* a *upload vulnerabilities*. Samozrejme je možné nástroj *WebCop* rozšíriť o ďalšie typy zraniteľností webových aplikácií.

Ako nevýhodu frameworku *w3af* oproti nástroju *WebCop* vidím to, že užívateľ po nakonfigurovaní testov nemá prakticky žiaden prehľad o tom, aké HTTP requesty tento testovací nástroj skutočne vyvolá. V nástroji *WebCop* je z užívateľskej konfigurácie približná štruktúra HTTP dotazov zrejmá.

S tým súvisí ďalšia výhoda nástroja *WebCop* - konfigurovateľnosť na úrovni jednotlivých testov. Užívateľ *w3af* napríklad nemá možnosť definovať *SQL*

/ *XSS injection* vektory, ktoré sa pri testovaní majú použiť. To znamená aj to, že ak chce užívateľ do nástroja pridať nový XSS vektor, musí upraviť zdrojový kód príslušného pluginu.

Ďalší rozdiel vidím v spôsobe testovania *CSRF* zraniteľností. Podľa komentárov v zdrojových kódoch príslušného pluginu sa dá zistiť, že nástroj *w3af* odošle formulár, a v prípade že webová aplikácia vráti perzistentné cookie, hodnotí test za úspešný. Ja takýto prístup hodnotím za nebezpečný, hlavne v prípade ak používateľ netestuje webovú aplikáciu, ktorej je sám aj administrátorom. Napríklad otestovanie aplikácie typu internet banking takýmto spôsobom by mohlo mať pre užívateľa katastrofálne následky. Takisto by sa dalo polemizovať o tom, nakoľko je odoslanie perzistentného cookie indikátorom úspešného prevedenia CSRF útoku. Ja som v nástroji *WebCop* zvolil opatrnejší prístup a to tým že testované formuláre na CSRF zraniteľnosti vôbec neodosielam, len ich analyzujem na prítomnosť náhodných tokenov.

6.2 Wapiti

Spoločnou črtou nástroja *Wapiti*, projektu *w3af* a nástroja *WebCop* je, že žiaden z týchto nástrojov nezávisí na externej databáze nejakých známych zraniteľností, ale tieto zraniteľnosti sami vyhľadávajú. Podobne ako projekt *w3af*, aj *Wapiti* je napísany v Pythone, čo so sebou prináša určitú platformovú nezávislosť. Na druhej strane, podobne ako v predchadzajúcom prípade, ani tento nástroj neumožňuje konfigurovať ako sa majú prevádzať jednotlivé testy. Tým myslím že užívateľ nemá kontrolu napríklad nad XSS vektormi, ktoré tento nástroj používa.

Rozsahom pokrytých zraniteľností je tento projekt podobný nástroju *WebCop*. Skener *Wapiti* síce neumožňuje detekciu zraniteľností *CSRF*, ale automatická detekcia tejto zraniteľnosti je problematická a ani moje riešenie, ani riešenie použité vo frameworku *w3af* určite nie je stopercentné.

6.3 Nikto

Základný rozdiel medzi nástrojom *WebCop* a *Nikto* je ten, že *Nikto* je skener webových serverov, nie konkrétnych webových aplikácií. S tým súvisí spôsob, ako *Nikto* vyhľadáva konkrétne zraniteľnosti - udržiava si databázu známych bezpečnostných chýb a testy prevádza vždy na konkrétnu zraniteľnosť v tejto databáze. Tento nástroj teda samostatne nevyhľadáva chyby vo webových aplikáciach, dokonca ani neprehľadáva štruktúru prítomných webových aplikácií.

6.4 Nessus

Keďže nástroj *Nessus* je určený na komplexný audit bezpečnosti počítačových sietí, má zmysel v tomto porovnaní sa sústrediť iba na tú časť funkcionality nástroja ktorá priamo súvisí s webom. *Nessus* sa dá použiť ako skener webových serverov, dokonca tento nástroj sa dá priamo prepojiť s nástrojom *Nikto*. Okrem takzvaného *signature based* testovania webových zraniteľností podľa svojej databázy chýb ale tento nástroj ponúka aj možnosti na samostatné vyhľadávanie bezpečnostných chýb vo webových aplikáciach.

Ako v predchádzajúcich nástrojoch aj tu však absentuje možnosť nakonfigurovania konkrétnych attack vektorov ktoré sa pri testovaní použijú. A podobne ako v nástroji *Wapiti*, chýba tu možnosť samostatného odhaľovania *CSRF* zraniteľností.

Kapitola 7

Záver

Nástroj *WebCop*, ktorý som v rámci tejto práce vytvoril, si kladie za cieľ pomôcť vývojárom webových aplikácií zautomatizovať a tým urýchliť proces vyhľadávania bežných bezpečnostných zraniteľností vo webových aplikáciach. Od užívateľa tohto nástroja sa pritom predpokladá znalosť princípu zraniteľností na ktoré danú webovú aplikáciu testuje. Konfiguráciou daných testov totiž užívateľ priamo ovplyvňuje ako sa samotný test vykoná a za akých podmienok test vyhodnotí prítomnosť danej zraniteľnosti pozitívne.

Nástroj *WebCop* je vytvorený ako Open Source projekt. Je hostovaný na serveri *sourceforge.net* s názvom projektu *webCop* pod licenciou *GNU General Public Licence (GPL)*.

Ďalší priestor pri rožširovaní tohto nástroja vidím predovšetkým v rozšírení pokrytia typov webových zraniteľností, ktorých prítomnosť nástroj umožňuje detekovať. Jednoducho by sa napríklad dala doplniť funkcionalita na detekciu zabudnutých backup súborov vo webových aplikáciach. Ďalšie užitočné rozšírenie tohto nástroja vidím v pridaní podpory protokolu HTTP/TLS, najjednoduchšie pomocou využitia nejakej third-party knižnice.

Appendix A

Obsah priloženého CD

Súčasťou práce je na CD priložená webová aplikácia, ktorá slúži na demonštrovanie funkčnosti nástroja. Táto aplikácia je hostovaná aj na internete na serveri *webzdarma.cz*. Nástroj takisto obsahuje už vyplnené konfiguračné súbory, ktoré sú nastavené tak že po skompilovaní a spustení tohto skenera webových zraniteľností sa začne testovať táto modelová webová aplikácia dostupná na adrese `http://profinzer.zaridi.to/`. Na CD je ďalej priložená programátorská dokumentácia vygenerovaná nástrojom *Doxygen*.

Literatúra

[1] A. Agarwwal, D. Belluci, A. Coronel, S. Di Paola, G. Fedon, A. Goodman, Ch. Heinrich, K. Horvath, G. Ingrosso, R. S. Liverani, A. Kuza, P. Luptak, F. Mavituna, M. Mella, M. Meucci, M. Morana, A. Parata, C. Su, H. S. Sureddy, M. Roxberry, A. Van der Stock. "The OWASP Testing Guide 3.0" Dec 2008 <http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf>

[2] Andrew van der Stock, Jeff Williams, Dave Wichers. "OWASP Top 10 project" 2007 <http://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf>

[3] Steve Christey, Robert A. Martin. "Vulnerability Type Distributions in CVE"May 22 2007 <http://cwe.mitre.org/documents/vuln-trends/index.html>

[4] ZeQ3uL && JabAv0C. "LFI to RCE Exploit with Perl Script" 07 Dec 2007 <http://www.milw0rm.com/papers/260>

[5] RSnake. "XSS (Cross Site Scripting) Cheat Sheet " 2008 <http://ha.ckers.org/xss.html>

[6] cOndemned. "CodeDB (list.php lang) Local File Inclusion Vulnerability" July 14 2008 <http://www.milw0rm.com/exploits/6071>

[7] Stephen Schwing, John Lampe. "Web Application Security Testing with the Security Center and Nessus" Feb 2007 <www.nessus.org/whitepapers/sec_test_sc3_nessus.pdf>

[8] Billy (BK) Rios. "SUN Fixes GIFARs" Dec 17 2008 <http://xs-sniper.com/blog/2008/12/17/sun-fixes-gifars/>

[9] Andrés Riancho. <http://w3af.sourceforge.net/>

[10] Nicolas Surribas. <http://wapiti.sourceforge.net/>

[11] Chris Sullo. <http://www.cirt.net/nikto2>

[12] Tenable Network Security, Inc. <http://www.nessus.org/nessus/>